Introduction to GraphQL

Christopher Bartling

November 6, 2019

REST

- Data requirements are dictated by the server-side
  - Multiple requests to fetch object graphs
- Multiple views of the same REST endpoint
  - Compact vs full views
- API evolution via versioned endpoints
- Weakly-typed endpoints

REST issues

- Over-fetching superfluous data
- Multiple requests to materialize resource graphs
  - Client takes responsibility for orchestrating data fetching and assembling data graph
- Payloads tend to grow over time, resulting in over-fetching
- Code duplication can occur when supporting multiple versions (server-side)
- Client changes when new versions of endpoints are rolled out

GraphQL Principles

- Hierarchical, graph-oriented

GraphQL Principles

- Hierarchical, graph-oriented
- Product-centric data requirements

GraphQL Principles

- Hierarchical, graph-oriented
- Product-centric data requirements
- Client-specified queries

GraphQL Principles

- Hierarchical, graph-oriented
- Product-centric data requirements
- Client-specified queries
- Backwards compatible

GraphQL Principles

- Hierarchical, graph-oriented
- Product-centric data requirements
- Client-specified queries
- Backwards compatible
- Application-layer protocol

GraphQL Principles

- Hierarchical, graph-oriented
- Product-centric data requirements
- Client-specified queries
- Backwards compatible
- Application-layer protocol
- Strongly-typed

GraphQL Principles

- Hierarchical, graph-oriented
- Product-centric data requirements
- Client-specified queries
- Backwards compatible
- Application-layer protocol
- Strongly-typed
- Introspective

Declarative Data Fetching

- Query language to satisfy data requirements for the client

Declarative Data Fetching

- Query language to satisfy data requirements for the client
- Client defines what will be included in the query response, not the server

Declarative Data Fetching

- Query language to satisfy data requirements for the client
- Client defines what will be included in the query response, not the server
- Data requirements are specified as a hierarchy of fields

Declarative Data Fetching

- Query language to satisfy data requirements for the client
- Client defines what will be included in the query response, not the server
- Data requirements are specified as a hierarchy of fields
- Avoid calling multiple endpoints

Declarative Data Fetching

- Query language to satisfy data requirements for the client
- Client defines what will be included in the query response, not the server
- Data requirements are specified as a hierarchy of fields
- Avoid calling multiple endpoints
- Avoid aggregating data manually

Declarative Data Fetching

- Query language to satisfy data requirements for the client
- Client defines what will be included in the query response, not the server
- Data requirements are specified as a hierarchy of fields
- Avoid calling multiple endpoints
- Avoid aggregating data manually
- Avoid over-fetching and under-fetching data

Developer Experience

- GraphQL delivers better developer experience with. . .
  - a self describing API which can be introspected by tooling and tooling can then validate against schema
  - query and mutation input validation
  - query facilities that aggregate data on the server-side
  - no need for versioning; query resolvers are independent of one another

Performance Improvements

- GraphQL delivers better performance by...
  - multiple independent queries can be sent in a single HTTP request
  - reducing the number of requests for a data graph
  - aggregating the data graph on the server-side, potentially across service boundaries
  - only sending the data fields requested

Schema Definition Language (SDL)

- Strong type system
- Type system == GOOD!
- Type language: Schema Definition Language (SDL)
- GraphQL schema can be introspected by tools and runtimes

User-defined Scalars

```
scalar uuid
scalar timestamp
scalar secureUrl
```

Object Types and Fields

```
type Actor {
  id: uuid!
  firstName: String!
  lastName: String!
}
```

User-defined Object Type Field

```
type ActorAggregateFields {
  count(columns: [ActorSelectColumn!],
        distinct: Boolean): Int
  max: ActorMaxFields
  min: ActorMinFields
}
```

Enumerations

```
enum ConflictAction {
  ignore
  update
}
```

Lists and Non-null

```
type ActorsAggregate {
  aggregate: ActorAggregateFields
  nodes: [Actor!]!
}
```

Interfaces

```
interface Person {
  id: ID!
  firstName: String!
  lastName: String!
}

type DraftProspect implements Person {
  id: ID!
  firstName: String!
  lastName: String!
  position: FootballPosition!
}
```

Union

```
union SearchResult = Human | Droid | Starship
```

Input Types

```
input ReviewInput {
  stars: Int!
  commentary: String
}
```

GraphQL Queries

- Queries retrieve data

GraphQL Queries

- Queries retrieve data
- Query structure mimics data structure in response

Query type

```
type Query {
  hero(episode: Episode): Character
  droid(id: ID!): Droid
}
```

Query example

```
query {
  hero {
    name
  }
  droid(id: "2000") {
    name
  }
}
```

GraphQL Mutations

- Mutations create, update, or remove data

GraphQL Mutations

- Mutations create, update, or remove data
- Typically use input types for specifying a grouping of fields

Mutation type

```
type Mutation {
  addBook(input: AddBookInput!): Book
  removeBook(id: ID!): Boolean
}
```

Mutation example

```
mutation AddBook($input: AddBookInput!) {
  addBook(input: $input) {
    id
  }
}
```

- Input is bound to variables in client

GraphQL Subscriptions

- Server-sent events

GraphQL Subscriptions

- Server-sent events
- Asynchronous

GraphQL Subscriptions

- Server-sent events
- Asynchronous
- Communication through WebSockets

GraphQL Subscriptions

- Server-sent events
- Asynchronous
- Communication through WebSockets
- Server-side implementation dependent on platform

Subscription type

```
type Subscription {
  commentAdded(input: CommentAddedSubscribeInput!): Comment
}
```

Subscription type

```
subscription CommentAddedSubscription(
    $input: CommentAddedSubscribeInput!
  ) {
  commentAddedSubscribe(input: $input) {
    comment {
      id
      commentText
      commenter {id, firstName, lastName}
    }
  }
}
```

Schema declaration

```
schema {
  query: Query
  mutation: Mutation
  subscription: Subscription
}
```

Server implementations

- Apollo Server (Node.js)
  - https://www.apollographql.com/docs/apollo-server/
- GraphQL Ruby (Ruby/RoR)
  - https://graphql-ruby.org/
- GraphQL Java (Java)
  - https://www.graphql-java.com/
- GraphQL .NET (.NET)
  - https://graphql-dotnet.github.io/

Clients

- Apollo Client (JavaScript)
  - https://www.apollographql.com/docs/react/
- Relay Modern (JavaScript/React)
  - (https://relay.dev/)
- Apollo Client (iOS)
  - https://www.apollographql.com/docs/ios/
- Apollo Client (Android)
  - https://github.com/apollographql/apollo-android

Tools

- graphql-tools
  - https://www.apollographql.com/docs/graphql-tools/
- Insomnia
  - https://insomnia.rest/graphql/
- Altair
  - https://altair.sirmuel.design/
- Postman
  - https://learning.getpostman.com/docs/postman/sending-api-requests/graphql/

Recommended reading

- Principled GraphQL
  - https://principledgraphql.com/
- Production Ready GraphQL
  - https://productionreadygraphql.com/
- The GraphQL Guide
  - https://graphql.guide/
- Awesome list of GraphQL and Relay
  - https://github.com/chentsulin/awesome-graphql

This presentation

- https://github.com/cebartling/graphql-introduction

Literature Cited

- https://reactjs.org/blog/2015/05/01/graphql-introduction.html
- https://www.upwork.com/hiring/development/why-facebooks-graphql-language-should-be-on-your-radar/
- https://www.youtube.com/watch?v=pJamhW2xPYw&amp=&feature
- https://speakerdeck.com/dschafer/graphql-client-driven-development?slide=61

Literature Cited

- https://crystallize.com/blog/better-developer-experience-with-graphql
- https://graphql.org/blog/subscriptions-in-graphql-and-relay/
- https://blog.echobind.com/8-business-reasons-to-adopt-graphql-and-apollo-server-c2e0e9655310